Three Dimensions of Process Improvement

Part II: The Personal Process

Watts S. Humphrey
Software Engineering Institute

Part I of this article (CROSSTALK, February 1998) described the Capability Maturity Model®, why it was developed, and how it can help organizations improve their performance. Part II addresses the Personal Software Process (PSP)℠, which shows engineers how to perform their tasks in an effective and professional way. In the final analysis, to have high-performance software organizations, you must have high-performance software engineers working on high-performance software teams. The objective of the PSP is to show software engineers how to use process principles in their work. Part III of this article (April 1998 issue of CROSSTALK) describes the Team Software Process, which shows integrated product teams how to consistently produce quality products under aggressive schedules and for their planned costs.

Moving from "What" to "How"

Although the Capability Maturity Model (CMM) provides a powerful improvement framework, its focus is necessarily on "what" organizations should do and not "how" they should do it. This is a direct result of the CMM's original motivation to support the Department of Defense acquisition community. We knew management should set goals for their software work but we also knew that there were many ways to accomplish these goals. Above all, we knew no one was smart enough to define how to manage all software organizations. We thus kept the CMM focus on goals, with only generalized examples of the practices the goals implied.

As organizations used the CMM, many had trouble applying the CMM principles. In small groups, for example, it is not generally possible to have dedicated process specialists, so every engineer must participate at least part time in process improvement. We kept describing to engineers *what* they ought to do and they kept asking us *how* to do it. Not only did this imply a need for much greater process detail, it also required that we deal more explicitly with the real practices of development engineers. We needed to show them precisely how to apply the CMM process principles.

Because software development is a rich and sophisticated process, we realized a single set of cookbook methods would not be adequate. We thus chose to deal with fundamental process principles and to show engineers how to define, measure, and improve their personal work. The key is to recognize that all engineers are different and that each must know how to tune their practices to produce the most personal benefit.

Changing Engineers' Practices

Improvement requires change, and changing the behavior of software engineers is a nontrivial problem. The reasons for this explain why process improvement is difficult and illustrate the logic for the PSP.

The problems related to improving the personal practices of software engineers have long interested me, so after I had been at the Software Engineering Institute (SEI) for several years, I looked for someone else to lead the CMM work so I could address this issue. I decided to first demonstrate how process improvement principles could be applied to the work of individual engineers. Over the next several years, I wrote 62 small to moderate-sized programs as I developed as close to a Level 5 personal process as I could devise.

The results were amazing. I became more productive, the quality of my work improved sharply, and I could make accurate personal plans. The next step was to demonstrate the effectiveness of these methods for others. I first tried meeting with engineering groups to describe what I had done and to get them to try it. Despite management support, this was a dismal failure. One laboratory manager even told his people that it was more important for them to use these methods than to meet their project schedules. The engineers all said they would do so, but none of them did. The question was why not?

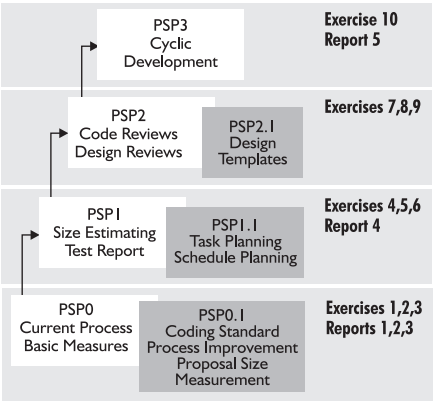


Figure 1. *The PSP process evolution.*

A Question of Conviction

Software engineers develop their personal practices when they first learn to write programs. Since they are given little or no professional guidance on how to do the work, most engineers start off with exceedingly poor personal practices. As they gain experience, some engineers may change and improve their practices, but many do not. In general, however, the highly varied ways in which individual software engineers work are rarely based on a sound analysis of available methods and practices.

Engineers are understandably skeptical about changes to their work habits; although they may be willing to make a few minor changes, they will generally

The SEI's work is supported by the Department of Defense. Capability Maturity Model and CMM are registered with the U.S. Patent and Trademark Office. Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

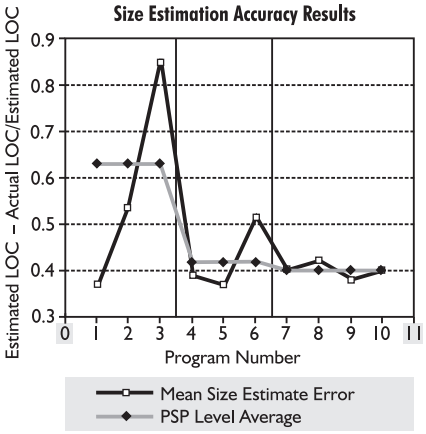


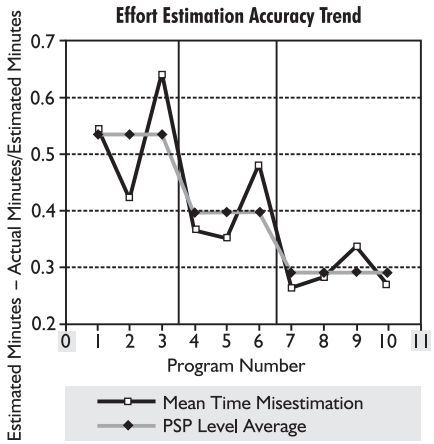
Figure 2. *Size estimation results.*

stick fairly closely to what has worked for them in the past until they are convinced a new method will be more effective. This, however, is a chicken-and-egg problem: engineers only believe new methods work after they use them and see the results, but they will not use the methods until they believe they work.

The Personal Software Process

Given all this, how could we possibly convince engineers that a new method would work for them? The only way we could think of to change this behavior was with a major intervention. We had to directly expose the engineers to the new way of working. We thus decided to remove them from their day-to-day environment and put them through a rigorous training course. As shown in Figure 1, the engineers follow prescribed methods, represented as levels PSP0 through PSP3, and write a defined set of 10 programming exercises and five reports [1]. With each exercise, they are gradually introduced to various advanced software engineering methods. By measuring their own performance, the engineers can see the effect of these methods on their work.

Figure 3. *Effort estimation results.*



Figures 2 through 5 show some of the benefits engineers experience [2, 3]. Figure 2 shows the average reduction in size-estimating error for nearly 300 engineers who took the PSP course and provided data to the SEI. Their size-estimating error at the beginning of the course is indicated at the left of the chart, and their error at the end of the course is shown at the right. This shows

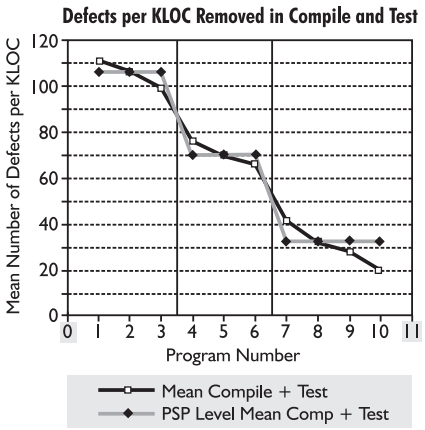


Figure 4. *Quality results.*

that size-estimating errors averaged 63 percent with PSP0 (the first three programs) and 40 percent for PSP2 and PSP3 (Programs 7, 8, 9, and 10). Note that the PSP introduces a disciplined estimating method (Proxy-Based Estimating) with PSP1 (Program 4) [1].

Similarly, for time estimating, Figure 3 shows an improvement from a 55 percent error to a 27 percent error or a factor of about two. As shown in Figure 4, the improvement in compile and test defects is most dramatic. From PSP0 to PSP3, the engineers' compile and test defects dropped from 110 defects per 1,000 lines of code (KLOC) to 20 defects per KLOC, or over five times. Figure 5 shows that even with their greatly improved planning and quality performance, the engineers' lines of code productivity was more or less constant.

Perhaps the most impressive PSP change is in the way the engineers spend their time. With Program 1, as shown in Figure 6, this group of nearly 300 engi-

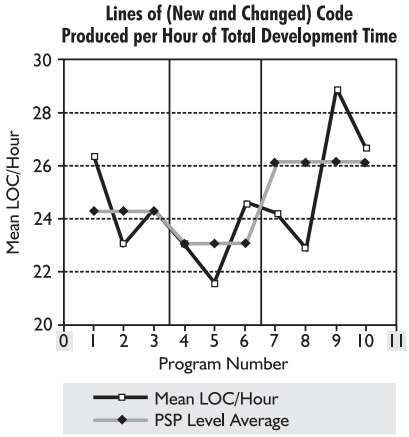


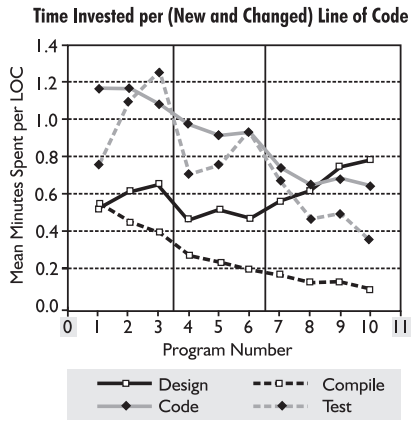
Figure 5. *Productivity results.*

neers spent on average less time designing their programs than they did on any other task. They even spent more time compiling than designing. At the end of the course, they spent more time designing than in any other technical activity. We have been trying to get software engineers to do this for years. Until they can experience the benefits of more thorough designs, they will likely continue to concentrate on coding, compiling, and testing.

Industrial Results with the PSP

A growing number of organizations are using the PSP, such as Baan, Boeing, Motorola, and Teradyne. Data from some early users clearly demonstrate the benefits of PSP training [4]. Figure 7 shows data from a team at Advanced Information Services (AIS) in Peoria, Ill. They were PSP trained in the middle of their project. The three bars on the left of the chart show the engineers' time estimates for the weeks it would take them to develop the first three compo-

Figure 6. *Effort distribution results.*



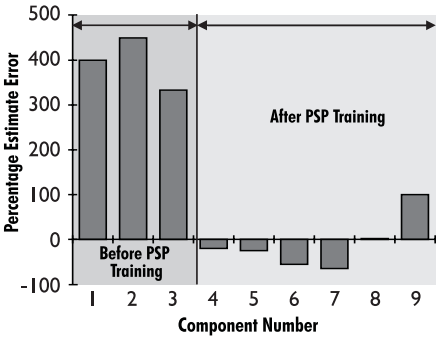


Figure 7. *Schedule estimating error.*

nents. For Component 1, for example, the original estimate was four weeks, but the job took 20 weeks. Their average estimating error was 394 percent. After PSP training, these same engineers completed the remaining six components. As shown on the right, their average estimating error was -10.6 percent. The original estimate for Component 8, for example, was 14 weeks and the work was completed in 14 weeks.

Table 1 shows acceptance test data on products from one group of AIS engineers. Before PSP training, they had a substantial number of acceptance test defects and their products were uniformly late. After PSP training, the next product was nearly on schedule, and it had only one acceptance test defect. Table 2 shows the savings in system testing time for nine PSP projects. At the top of the chart, system test time is shown for several products that were completed before PSP training. At the bottom, system test time is shown for products the same AIS engineers completed after PSP training. Note that A1 and A2 are two parts of the same product, so testing for them was done together in one and one-half months.

Table 1. *Acceptance test improvement.*

Not Using PSP	KLOC	Months Late	Acceptance Test Defects
1	24.6	9	N/A
2	20.8	4	168
3	19.9	3	21
4	13.4	8+	53
5	4.5	8+	25
Using PSP			
1	22.9	1	1

Introducing the PSP

Although the PSP can be introduced quickly, it must also be done properly. First, the engineers need to be trained by a qualified PSP instructor. The SEI trains and authorizes PSP instructors and provides limited on-site PSP training. There is also a growing number of SEI-trained PSP instructors who offer commercial PSP training (see http://www.sei.cmu.edu).

The second important step in PSP introduction is to train in groups or teams. When organizations ask for volunteers for PSP training, they get a sparse sprinkling of PSP skills that will generally have no impact on the performance of any project.

System test time before PSP training

Project	Size	Test Time
A1	15,800 LOC	1.5 months
C	19 requirements	3 test cycles
D	30 requirements	2 months
H	30 requirements	2 months

System test time after PSP training

Project	Size	Test Time
A2	11,700 LOC	1.5 months
B	24 requirements	5 days
E	2,300 LOC	2 days
F	1,400 LOC	4 days
G	6,200 LOC	4 days
I	13,300 LOC	2 days

Table 2. *System test time savings.*

Third, effective PSP introduction requires strong management support. This, in turn, requires that management understand the PSP, know how to support their workers once they are trained, and regularly monitor their performance. Without proper management attention, many engineers gradually slip back into their old habits. The problem is that software engineers, like most professionals, find it difficult to consistently do disciplined work when nobody notices or cares. Software engineers need regular coaching and support to sustain high levels of personal performance.

The final issue is that even when a team of engineers are all PSP trained and properly supported, they still have to figure out how to combine their personal processes into an overall team process. We have found this to be a problem even at higher CMM levels. These are the reasons we are developing the Team Software Process (TSP).

Part III of this article, which describes what the TSP is and how it helps teams to work more effectively, will appear in the April 1998 issue of *CROSSTALK*. Although the TSP is still in development, early industrial experience demonstrates that it can substantially improve the performance of integrated product teams. ◆

About the Author

Watts S. Humphrey is a fellow at the SEI of Carnegie Mellon University, which he joined in 1986. At the SEI, he established the Process Program, led initial development of the CMM, introduced the concepts of Software Process Assessment and Software Capability Evaluation, and most recently, the PSP and TSP. Prior to joining the SEI, he spent 27 years with IBM in various technical executive positions, including management of all IBM commercial software development and director of programming quality and process. He has master's degrees in physics from the Illinois Institute of Technology and in business administration from the University of Chicago.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Voice: 412-268-6379
E-mail: watts@sei.cmu.edu

References

1. Humphrey, W.S., *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
2. Hayes, Will, "The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers," CMU/SEI-97-TR-001.
3. Humphrey, W.S., "Using a Defined and Measured Personal Software Process," *IEEE Software*, May 1996.
4. Ferguson, Pat, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya, "Introducing the Personal Software Process: Three Industry Case Studies," *IEEE Computer*, Vol. 30, No. 5, May 1997, pp. 24-31.